Physical database design

Elmasri/Navathe ch 17 and 18
Padron-McCarthy/Risch ch 21 and 22

DATABASE DESIGN I - 1DL300 Fall 2012
Sobhan Badiozamany
Silvia Stefanova
Department of Information Technology, Uppsala University

Select *
from employee e
where e.manager=10;

Query parser and optimizer

Execution plan: An storage/retrieval program

Physical database

The physical database is a collection of stored records that have been organized in files on the hard disk.

Data files

Index Files

Hard Disk System
Physical database design

Elmasri/Navathe ch 17 and 18
Padron-McCarthy/Risch ch 21 and 22

DATABASE DESIGN I - 1DL300 Fall 2012
Sobhan Badizamany
Silvia Stefanova
Department of Information Technology, Uppsala University

Select *
from employee e
where e.manager=10;

Physical database

The physical database is a collection of stored records that have been organized in files on the hard disk.

Query parser and optimizer
Execution plan: An storage/retrieval program

Index Files

Data files

Hard Disk System
Select *
from employee e
where e.manager=10;

The physical database is a collection of stored records that have been organized in files on the hard disk.
Hard Disk System

Buffer:
portion of main memory available to store copies of disk blocks.

The DBMS engine calls the buffer manager when it needs a block from disk:
- The DBMS is given the address of the block in the main memory, if it is already present in the buffer.
- If the block is not in the buffer:
  - the buffer manager allocates space in the buffer for the block.
  - If buffer is full, it has to throw out some other block.
- The block that is thrown out is written back to disk, only if it was modified since the most recent time that it was written (flushed) from the disk.
- Once space is allocated in the buffer, the buffer manager reads the block from the disk to the buffer, and passes the address of the block in main memory to the requester.

Block transfer is slow
The goal:
- Minimize the number of blocks transferred.
- Minimize the arms movements.

The subsystem responsible for allocating buffer space in main memory:

- Database system tries to minimize the number of block transfers between the disk and main memory.
- We can reduce the number of disk accesses by keeping as many blocks as possible in main memory.

Disks

Data in Blocks
Disks

- Preferred secondary storage device for high storage capacity and low cost.
- Data stored as magnetized areas on magnetic disk surfaces.

- Disks are divided into concentric circular tracks on each disk surface.

  A track is divided into smaller blocks or sectors:
  - The block size $B$ is fixed for each system.
    - Typical block sizes range from $B=512$ bytes to $B=4096$ bytes.
  - Whole blocks are transferred between disk and main memory for processing.

A read-write head moves to the track that contains the block to be transferred.

- Disk rotation moves the block under the read-write head for reading or writing.

Reading or writing a disk block is time consuming because of:

  - the seek time: the time it takes to move the arm.
  - rotational delay: the time it takes for the disk to rotate such that the head reaches the desired sector.
Disks

- Preferred secondary storage device for high storage capacity and low cost.
- Data stored as magnetized areas on magnetic disk surfaces.

Figure 17.2
Different sector organizations on disk. (a) Sectors subtending a fixed angle. (b) Sectors maintaining a uniform recording density.
Disks are divided into concentric circular tracks on each disk surface.

A track is divided into smaller blocks or sectors:
- The block size $B$ is fixed for each system.
  - Typical block sizes range from $B=512$ bytes to $B=4096$ bytes.
- Whole blocks are transferred between disk and main memory for processing.
A read-write head moves to the track that contains the block to be transferred.

- Disk rotation moves the block under the read-write head for reading or writing.

Reading or writing a disk block is time consuming...
A read-write head moves to the track that contains the block to be transferred.

- Disk rotation moves the block under the read-write head for reading or writing.

Reading or writing a disk block is time consuming because of:

- the seek time: the time it takes to move the arm.
- rotational delay: the time it takes for the disk to rotate such that the head reaches the desired sector.
The DBMS engine calls the buffer manager when it needs a block from disk.
- The DBMS is given the address of the block in the main memory. If it is already present in the buffer.
- If the block is not in the buffer:
  - the buffer manager allocates space in the buffer for the block.
  - If buffer is full, it has to throw out some other block.
- The block that is thrown out is written back to disk only if it was modified since the most recent time that it was written to/from the disk.
- Once space is allocated in the buffer, the buffer manager reads the block from the disk to the buffer and passes the address of the block in main memory to the requester.

Block transfer is slow
The goal:
- Minimize the number of blocks transferred.
- Minimize the arms movements.

Buffer:
portion of main memory available to store copies of disk blocks.

The subsystem responsible for allocating buffer space in main memory.

- Database system tries to minimize the number of block transfers between the disk and main memory.
- We can reduce the number of disk accesses by keeping as many blocks as possible in main memory.
The DBMS engine calls the buffer manager when it needs a block from disk.

- The DBMS is given the address of the block in the main memory, if it is already present in the buffer.

- If the block is not in the buffer:
  - the buffer manager allocates space in the buffer for the block.
  - if buffer is full, it has to through out some other block.

- The block that is thrown out is written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk.
  - Once space is allocated in the buffer, the buffer manager reads in the block from the disk to the buffer, and passes the address of the block in main memory to the requester.
memory.

- Database system tries to minimize the number of block transfers between the disk and main memory.
- We can reduce the number of disk accesses by keeping as many blocks as possible in main memory.
The physical database is a collection of stored records that have been organized in files on the hard disk.

Records are used for physical storage of:

- Tuples, where each attribute in the tuple is stored as a field.
- Objects in object-oriented databases.

Records contain fields which have values of a particular type.
## Data files

### File header
- Contains information that is needed for record access.
- Block addresses
- Record format

<table>
<thead>
<tr>
<th>Block 0</th>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 4</td>
<td>Block 5</td>
<td>Block 6</td>
<td>Block 7</td>
</tr>
<tr>
<td>Block 8</td>
<td>Block 9</td>
<td>Block 10</td>
<td>Block 11</td>
</tr>
<tr>
<td>Block 12</td>
<td>Block 13</td>
<td>Block 14</td>
<td>Block 15</td>
</tr>
<tr>
<td>Block 16</td>
<td>Block 17</td>
<td>Block 18</td>
<td>Block 19</td>
</tr>
<tr>
<td>Block 20</td>
<td>Block 21</td>
<td>Block 22</td>
<td>Block 23</td>
</tr>
<tr>
<td>Block 24</td>
<td>Block 25</td>
<td>Block 26</td>
<td>Block 27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block 0</th>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 4</td>
<td>Block 5</td>
<td>Block 6</td>
<td>Block 7</td>
</tr>
<tr>
<td>Block 8</td>
<td>Block 9</td>
<td>Block 10</td>
<td>Block 11</td>
</tr>
<tr>
<td>Block 12</td>
<td>Block 13</td>
<td>Block 14</td>
<td>Block 15</td>
</tr>
<tr>
<td>Block 16</td>
<td>Block 17</td>
<td>Block 18</td>
<td>Block 19</td>
</tr>
<tr>
<td>Block 20</td>
<td>Block 21</td>
<td>Block 22</td>
<td>Block 23</td>
</tr>
<tr>
<td>Block 24</td>
<td>Block 25</td>
<td>Block 26</td>
<td>Block 27</td>
</tr>
</tbody>
</table>

Similarly, on
File header

- Contains information that is needed for record access:
  - Block addresses
  - record format etc.

<table>
<thead>
<tr>
<th>Record 0</th>
<th>Record 1</th>
<th>Record 2</th>
<th>Record 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record 4</td>
<td>Record 5</td>
<td>Record 6</td>
<td>Record 7</td>
</tr>
<tr>
<td>Record 8</td>
<td>Record 9</td>
<td>Record 10</td>
<td>Record 11</td>
</tr>
<tr>
<td>Record 12</td>
<td>Record 13</td>
<td>Record 14</td>
<td>Record 15</td>
</tr>
<tr>
<td>Record 16</td>
<td>Record 17</td>
<td>Record 18</td>
<td>Record 19</td>
</tr>
<tr>
<td>Record 20</td>
<td>Record 21</td>
<td>Record 22</td>
<td>Record 24</td>
</tr>
<tr>
<td>Record 25</td>
<td>Record 26</td>
<td>Record 27</td>
<td>Record 28</td>
</tr>
</tbody>
</table>

Block 0  Block 1  Block 2  Block 3  Block 4  Block 5  Block 6
Files with fixed length record.

For example, consider a file containing records of this type:

All fields have fixed length.

<table>
<thead>
<tr>
<th>Name</th>
<th>personnummer</th>
<th>Salary</th>
<th>Job_code</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anders, Andersson</td>
<td>861115-5874</td>
<td>30,000</td>
<td>F-34174</td>
<td>Fincance</td>
</tr>
</tbody>
</table>

Advantages:

- Easy to locate record \( r \), given its record number \( n \) and size of each record \( s \):
All fields have fixed length.

<table>
<thead>
<tr>
<th>Name</th>
<th>personnummer</th>
<th>Salary</th>
<th>Job_code</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anders, Andersson</td>
<td>861115-5874</td>
<td>30,000</td>
<td>F-34174</td>
<td>Fincance</td>
</tr>
</tbody>
</table>

Advantages:
- Easy to locate record $r$, given its record number $n$ and size of each record $s$:
  - $\text{position}(r) = n \times s$
- Easy to locate field values

Disadvantage:
- Space is wasted
mix of fixed and variable length records

<table>
<thead>
<tr>
<th>Name</th>
<th>personnummer</th>
<th>Salary</th>
<th>Job_code</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith, John</td>
<td>861115-5874</td>
<td>20,000</td>
<td></td>
<td>Information Technology</td>
</tr>
</tbody>
</table>

Field separator: 

Advantages:

- space is wasted, but only where NULL values occur
<table>
<thead>
<tr>
<th>Name</th>
<th>personnummer</th>
<th>Salary</th>
<th>Job_code</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith, John</td>
<td>861115-5874</td>
<td>20,000</td>
<td>Information Technology</td>
<td></td>
</tr>
</tbody>
</table>

Field separator:  

**Advantages:**

- space is wasted, but only where NULL values occur.

**Disadvantage:**

locating records and fields needs more processing:

- location of the salary field depends on the length of name in the record.
files with Variable-field records

| Name=Smith, John | Personnummer=861115-5874 | Department=Information Technology |

Field separator: | Record terminator: 

Advantages:

- Less space is wasted, specially if there are NULL values.
Field separator: ❌  
Record terminator: ☑️

**Advantages:**
- Less space is wasted, specially if there might be many NULL values.

**Disadvantage:**
- More processing is needed for:
  - locating records/fields.
  - Identifying NULL values
    - need to see the whole record to identify a NULL value
There might be left over space in the file blocks

Record 1

Record 2

Blocking factor

The highest number of records that can be contained in a block is called the block factor (here: 1.5) for the file of records.

\[ BF = \frac{RN}{BR} \]

where \( B \) is the record size and \( R \) the block size.

In this example, assume a block size \( B \) is 512 bytes, and record size \( R \) is 200 bytes.

- \( B / R = 512 / 200 = 2.56 \)
- Rounded off downwards gives BF = 2, i.e., we can store 2 records per block.

There is space, but only two records can fit, so part of the block is wasted.
Blocking factor

The highest number of records that can be contained in a block is called the block factor (here: bfr) for the file of records.

\[ \text{bfr} = \text{floor}[\frac{B}{R}] \]

where \( R \) is the record size and \( B \) the block size.

In this example, assume a block size \( B \) is 512 bytes, and record size \( R \) is 200 bytes.

- \( \frac{B}{R} = \frac{512}{200} = 2.56 \)
- Rounded off downwards gives \( \text{bfr} = 2 \), i.e. we can store 2 records per block.
Similarly, on file level:

A file with $r$ no. of records therefore require:

$$b = \text{ceiling}\left[\frac{r}{\text{bfr}}\right]$$

blocks.

In our example, bfr=2, so a file with $r=605$ records needs $605/2=302.5 \sim 303$ blocks.

<table>
<thead>
<tr>
<th>File header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 0</td>
</tr>
<tr>
<td>Block 4</td>
</tr>
<tr>
<td>Block 8</td>
</tr>
<tr>
<td>Block 12</td>
</tr>
<tr>
<td>Block 16</td>
</tr>
<tr>
<td>Block 20</td>
</tr>
<tr>
<td>Block 24</td>
</tr>
</tbody>
</table>

Q: How much space is wasted?
File organization

- The database is stored as a collection of files. Each file is a sequence of records.
- Records can have constant (simplest) or variable length
- A file can store records of the same type (simplest) or of different type.
- Specific files can be used to store specific relations (simplest) or the same file can store different relations (maybe even the whole database).
Organization of records in files

- A record can be placed anywhere in the file where there is space.
- Records are stored in sequential order, based on the value of the search key of each record.
- A hash function is computed on some attribute of each record; the result specifies in which block of the file the record should be placed.

Heap files

- New records are added at the end of the file. Such an organization is called a heap file.
- Records are stored in a way that allows for efficient retrieval.
- Records can be dynamically resized as needed.

Ordered files

- Records are stored in the file in order according to the value of a certain field, in the figure according to name.
- This allows for efficient sequential access and searching based on the field.

External Hashing

- The file blocks are divided into N egal partions. Each partion is its own mini database.
- Records are inserted into the database in sequential order based on the hashing function.
- This allows for efficient search and retrieval based on the hash value.
A record can be placed anywhere in the file where there is space.
Heap files

- New records are added to the end of the file. Such an organization is called a heap file.
  - Suitable when we don’t know how data shall be used.
- Insert of a new record is very efficient.
- Search after a specific record is expensive (linear to the size).
- Delete of a record can be expensive (search - read into - delete - write back).
  - Instead of physically removing a record one can mark the record as deleted. Both methods require a periodically reorganization of the file.
- Modification of a record of variable length can be hard.
- Retrieval according to a certain order requires that the file must be sorted which is expensive.
store records in sequential order, based on the value of the search key of each record.

A hash function is computed on some attribute of each record; the result specifies a block of the file the record is stored in.
Ordered files

The records in the file are ordered according to the value of a certain field, in the figure according to name.

- Ordered retrieval very fast (no sorting needed).
- Next record in the order is found on the same block (except for the last record in the block)
- Search is fast (binary search - \( \log_2 b \))
- Insert and delete are expensive since the file must be kept sorted.
- Suitable for applications that require sequential processing of the entire file
- Need to reorganize the file from time to time to restore sequential order

• To make record insertions cheaper:
  - Create a temporary unsorted file a so called overflow file or transaction file
    (the main file is then called “the master file”)
  - Update the master file periodically in accordance with the transaction file.
  - These measures improve insertion time but search for records becomes more complicated.
• Ordered files are not used that often in databases.
  - Exception: when extra access paths are created, so called primary indexes.
According to the value of the name.

First for the last record in the file or transaction file.

Next of the entire file in sequential order.

Followed primary indexes.
records in the file are ordered according to the value of a certain field, in the figure according to name.

- Ordered retrieval very fast (no sorting needed).
- Next record in the order is found on the same block (except for the last record in the block).
- Search is fast (binary search - log2b).
- Insert and delete are expensive since the file must be kept sorted.
- Suitable for applications that require sequential processing of the entire file.
- Need to reorganize the file from time to time to restore sequential order.

- To make record insertions cheaper:
  - Create a temporary unsorted file a so called overflow file or transaction file (the main file is then called “the master file”).
  - Update the master file periodically in accordance with the transaction file.
  - These measures improve insertion time but search for records becomes more complicated.
Binary search

Watch between 0:50 to 01:32
the block

- Search is fast (binary search - \( \log_2 b \))
- Insert and delete are expensive since the file must be kept sorted.
- Suitable for applications that require sequential processing of the entire file
- Need to reorganize the file from time to time to restore sequential order

- To make record insertions cheaper:
  - Create a temporary unsorted file a so called overflow file or transaction file (the main file is then called "the master file")
  - Update the master file periodically in accordance with the transaction file.
  - These measures improve insertion time but search for records becomes more complicated.

- Ordered files are not used that often in databases.
  - Exception: when extra access paths are created, so called primary indexes.
A hash function is computed on some attribute of each record; the result specifies in which block of the file the record should be placed.
External Hashing

The file blocks are divided into $M$ equal-sized buckets, numbered bucket0, bucket1, ..., bucketM-1.

- Typically, a bucket corresponds to one (or a fixed number of) disk block.

One of the file fields is designated to be the hash key of the file. The record with hash key value $K$ is stored in bucket $i$, where $i=h(K)$, and $h$ is the hashing function.

Properties

- Search is very efficient on the hash key.
- Collisions occur when a new record hashes to a bucket that is already full.
- An overflow file is kept for storing such records.
- Overflow records that hash to each bucket can be linked together.

Example:
- Hash key:
  - employee number emp_num
- Number of Buckets $M=100$
- Hash function:
  - $h(emp_num) = emp_num \mod 100$

To find the block that contains the record for employee number 102, we should compute

$h(102)=102 \mod 100 = 2$

Figure 17.9
Matching bucket numbers to disk block addresses.
The file blocks are divided into M equal-sized buckets, numbered bucket0, bucket1, ..., bucketM-1.

- Typically, a bucket corresponds to one (or a fixed number of) disk block.

One of the file fields is designated to be the hash key of the file. The record with hash key value K is stored in bucket i, where i=h(K), and h is the hashing function.

**Example:**

Hash key:
- employee number emp_num

Number of Buckets M=100

Hash function:
Example:

Hash key:
  - employee number emp_num
Number of Buckets $M=100$
Hash function:
  - $h(\text{emp\_num}) = \text{emp\_num} \mod 100$

To find the block that contains the record for employee number 102 we should compute $h(102) = 102 \mod 100 = 2$.

Figure 17.9
Matching bucket numbers to disk block addresses.
The hash key value $K$ is stored in bucket $i$, where $i$ is determined by the hashing function.

The block that contains $\text{emp\_num}=102$ record.
Properties

- Search is very efficient on the hash key.
- Collisions occur when a new record hashes to a bucket that is already full.
  - An overflow file is kept for storing such records.
  - Overflow records that hash to each bucket can be linked together.
Indexing: Book example

Indexing in DBMS's

- To find records, one or several blocks transferred to (one or several buffers) into main memory. These blocks can then be searched to find the records that were sought.
- But which block?
  - If the address to the block containing the record is unknown one has to search through all block in the file (so called linear search).

If there was no index/table-of-content the student had to "scan" the whole book!

Index  Data
O o!
I need to start studying for the exam
:-(

Where is it in the book that
the relational algebra
is covered?
Table of contents

Chapter 1: introduction
...
...
Chapter 6: Formal Relational Languages: the Algebra and Calculus.
...
...
Index of key words

... 

... 

relational data model .... 55-80
relational algebra .......... 143-170

... 

...
Relational algebra

bla bla bla bla bla
bla bla bla bla bla
bla bla bla bla bla
bla bla bla bla bla
bla bla bla bla bla
bla bla bla bla 143
Indexing: Book example

Indexing in DBMS's

- To find records, one or several blocks transferred to (one or several buffers in) main memory. These blocks can then be searched to find the records that were sought.
- But which blocks?
- If the address to the block containing the record is unknown one has to search through all block in the file (so called linear search).

If there was no index/table-of-content the student had to "scan" the whole book!
Indexing in DBMS's

- To find records, one or several blocks transferred to (one or several buffers in) main memory. These blocks can then be searched to find the records that were sought.

- But which blocks?
  - If the address to the block containing the record is unknown one has to search through all block in the file (so called linear search).
Select *
from employee e
where e.manager=10;
Index Files

An index (or index file) is an extra file structure that is used to make the retrieval of records faster.

An index file consists of records (called index entries) of the form:  

Search key  |  pointer

Search key (or index field): Attribute or set of attributes (data fields) used to look up records in a file.

pointer (or address): The address of blocks that contain records with the given search key.

- These entries determine the physical address for records having a certain value in their index field.
- Index files are typically much smaller than the original file.
- The file that should be indexed is called the data file.
Index Files

An index (or index file) is an extra file structure that is used to make the retrieval of records faster.

An index file consists of records (called index entries) of the form:

```
Search key   pointer
```

Search key (or index field): Attribute or set of attributes (data fields) used to look up records in a file.

pointer (or address): The address of blocks that contain records with the given search key.

- These entries determine the physical address for records having a certain value in their index field.

Index files are typically much smaller than the original file.
An index file consists of records (called index entries) of the form: **Search key** | **pointer**

**Search key** (or index field): Attribute or set of attributes (data fields) used to look up records in a file.

**pointer** (or address): The address of blocks that contain records with the given search key.

- These entries determine the physical address for records having a certain value in their index field.
- Index files are typically much smaller than the original file.
- The file that should be indexed is called the **data file**.
Primary index

The data file is sorted on the primary key field.

Primary index - pros and cons

- Require much less space than the data file.
  - a) There is much fewer index records than records in the data file.
  - b) Every index record need less space (⇒ fewer memory blocks).
- Problem with insertion and deletion of records,
  - If anchor records are changed the index file must be updated.
Block anchor
primary key
value

<table>
<thead>
<tr>
<th>Aaron, Ed</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Adams, John</td>
<td></td>
</tr>
<tr>
<td>Alexander, Ed</td>
<td></td>
</tr>
<tr>
<td>Allen, Troy</td>
<td></td>
</tr>
<tr>
<td>Anderson, Zach</td>
<td></td>
</tr>
<tr>
<td>Arnold, Mack</td>
<td></td>
</tr>
</tbody>
</table>

Block pointer

- the same type as the ordering field (index field) for the data file.
- pointer to a block (block pointer)
The first record in each block is called the Anchor Record of the block.

**Figure 18.1**
Primary index on the ordering key field of the file shown in Figure 17.7.

- A primary index has one index record for each block in the data file, and therefore is called a **sparse index** (or “nondense index”)
- A **dense index** consists of one record for each record in the data file.
Primary index - pros and cons

- Require much less space than the data file.
  - a) There is much fewer index records than records in the data file.
  - b) Every index record need less space (⇒ fewer memory blocks).
- Problem with insertion and deletion of records.
  - If anchor records are changed the index file must be updated.
Cluster index

Cluster index is defined for files that are ordered according to a non-key field (the cluster field), i.e. several records in the data file can have the same value for the cluster field.

Figure 18.2
A clustering index on the Dept_number ordering non-key field of an EMPLOYEE file.
Cluster index is defined for files that are ordered according to a non-key field (the cluster field), i.e. several records in the data file can have the same value for the cluster field.
<table>
<thead>
<tr>
<th>Clustering field value</th>
<th>Block pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

- **Search key**: the same type as the Cluster field for the data file.
- **pointer**: pointer to a block (block pointer)
Secondary index

The data file is not sorted according to the index field.

There are two different cases:
1. The index field has unique values for each record (the figure).
2. Several records in the data file can have the same values for the index field (see Elmasri/Navathe Fig 18.5)
Secondary index

The data file is not sorted according to the index field.

There are two different cases:

1. The index field has unique values for each record (the case
   in Figure 18.4).
2. Several records in the data file can have the same values.
<table>
<thead>
<tr>
<th></th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
</tr>
</thead>
</table>

**Search key**

- the same type as the indexing field
- (any field in the data file)

**pointer**

- pointer to a block
- (block pointer)
Figure 18.4
A dense secondary index (with block pointers) on a nonordering key field of a file.

<table>
<thead>
<tr>
<th>Index file</th>
<th>Data file</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indexing field (secondary key field)</td>
</tr>
<tr>
<td>Index field value</td>
<td>Block pointer</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

There are two differences:
1. The index file contains values for each search key.
2. Several records can have the same value for the index field (see Fig 18.5)
Secondary index

The data file is not sorted according to the index field.

There are two different cases:
1. The index field has unique values for each record (the figure).
2. Several records in the data file can have the same values for the index field (see Elmasri/Navathe Fig 18.5)
Index evaluation metrics

Indexing techniques evaluated on basis of:

- Access types supported efficiently. E.g.,
  - records with a specified value in an attribute e.g. where salary=2000;
  - or records with an attribute value falling in a specified range of values.
  e.g. where salary between 2000 and 3000;
- Access time
- Insertion time
- Deletion time
- Space overhead
Indexing in SQL

CREATE INDEX index_name
ON table_name (column_name);

Creates an index on a table.
Duplicate values are allowed.

CREATE UNIQUE INDEX index_name
ON table_name (column_name);

Creates a unique index on a table.
Duplicate values are not allowed.
Physical database design

Elmasri/Navathe ch 17 and 18
Padron-McCarthy/Risch ch 21 and 22

DATABASE DESIGN I - 1DL300 Fall 2012
Sobhan Badizamany
Silvia Stefanova
Department of Information Technology, Uppsala University

Select *
from employee e
where e.manager=10;

Query parser and optimizer → Execution plan: An storage/retrieval program

The physical database is a collection of stored records that have been organized in files on the hard disk.